Dec. 26th, 2024 v.1.0

Exam - Algorithmics course 2024 Fall.

Period: From **Dec 26th** until **January 6th** (Monday) midnight (24:00).

Write on the first page of the PDF report

1) your **full name** and **study book number**,

2) **statement**, that you completed the exam **on your own (no other people were involved)**

3) Cite and report all materials and AI tools that you consulted and used in your solutions.

4) Keep track of time spent and report **the used hours for each task.** Report them.

Submission: via **courses.cs.ut.ee** - the **PDF** and code (zip, tar, gz, …)

Rules

Exam is personal, no collaborations or discussions are allowed with any other people - students, parents, colleagues, etc. But you can use Internet and AI assistants! However, you must describe which resources and assistants you have used.

Note that you may receive some of the points by properly describing theoretically working solutions even if you do not implement them. But you may not know if they would work without trying out, hence you might lose points. Describe concisely (briefly but sufficiently) how you went about solving the tasks; add high-level description of practical implementation and experimentation. Minimum no of points to pass the exam is 20. Exam was supposed to give up to 40 points. It is your choice to see how many you need or want to achieve.

Hint: There is no need to complete all programming tasks to perfection. Think carefully through and choose wisely. Same algorithm may or may not work on all test cases (describe if and why this may happen). Focus on the most obvious solutions and essential things first to get a baseline working for each task. Later you can improve, add layers of complexity or perfection. Goal is to demonstrate independent thinking. No need to optimize for maximal speed; but rather for clarity of thought. Measure code execution times but do not worry about whether you have achieved a new world record, fitting in exam time frame is more important than saving 20% time off the fast solution. Shorter code is preferred over the complex solutions.

Keep descriptions of your solutions of each task to ca 1-2 pages of textual explanations, pseudocode, discussion. Make sure to **add illustrations/visuals!** Ideally, the entire exam report should not exceed 10-12 pages. Upload also the code.

Dec. 26th, 2024 v.1.0

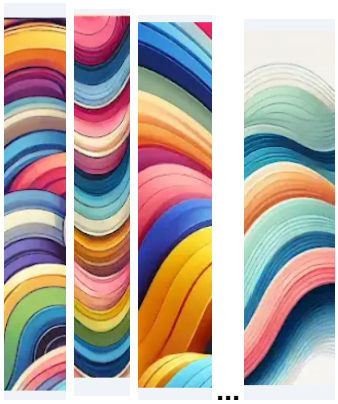
EX1 (15p): Find similar patterns of colors across multiple images.

In this folder there are many images with colourful designs.
https://drive.google.com/drive/folders/1CyYq8iqJti45w7MpuEWYD930dJ1fNSYY?usp=drive_link

Find similar patterns in images. Explore a couple of ideas and report their successes and failures. Report visually the top-10 “most similar” hits. Gradually increase complexity, e.g.:

- **Single vertical line (5p)**: One vertical line across all vertical lines in (same and) other images. Report lines from other images - use RGB as the guiding value. Ideally use cosine similarity between pixel intensities and Dynamic Time Warping for sequences. Discuss other potential similarity measures and test whether they would be also useful.
- **Band of lines (5p):** Repeat search for a multi-pixel band of lines (e.g. 2,5,10,20,50,...px wide bands?). What band-width delivers the most credible similarities? Compare bands, by for example converting bands into single lines or preferably adjusting the distance across multiple pixels in a band and applying DTW across them simultaneously.
- **Study the effect of resizing images (5p)**: Explore the effect of resizing/reshaping of underlying images - e.g. image 19 has been rescaled also by squeezing horizontally and vertically. You can of course resize/stretch images within your own code to test for how well the above single line or **band of lines** methods tolerate such stretching? Note that the width and length of the band depends on scaling of the image.
- Bonus (5p): search for (global or local) similarities not only in between vertical bands but more **arbitrary orientations**, across different images. Report those that are better matches than vertical bands only. Local similarity may be a “similar patch”.

Make sure to report results also visually. E.g. “searching from image X, I found most similar stretches from the following images: img, col_i .. col_j : (the example here is arbitrary, not selected for similarities). Note also that the images are arbitrary, you may generate more to have more meaningful proof of finding relevant similarities.



Original: image x, i..j;
Matches:

Image k, ik .. jk, distance
Image l, il .. jl, distance …

EX2 (15p): Finite automaton matching.

Given regular expressions over alphabet of three letters {a,b,c}:

(aba|ba|aaa)+

(ab | abc | aba)+ (ba (c+)) (cbb | bba | a)+

Start from the simpler one before moving to the other:

Match regular language using an NFA (5p)

Create the NFA based on Thompson and the Glushkov construction: Draw these automata on paper. Tabulate the transitions of automata in the report (this is also useful for debugging). No need to build a full parser (that would yield +5p bonus).

Synthesize sequences that belong to language represented by these automata. Report 10 examples, including **the shortest** and **some quite long** (ca 50-60 chars).

Make a random file / stream of many sequences over the alphabet of {a,b,c}; and inject the above 10 examples in these files, so that you know that there are at least these exact matches!

Match your NFA automata against that file; and report some more examples matches of the automata that were randomly there before injection of 10 examples. How frequently would a random sequence belong to the above two regular languages (estimate)?

NFA -> DFA -> minimal(DFA) (5 p)

Convert your automata into DFA. Draw DFA. **Match the DFA.**

Minimise the DFA; Draw and explain the minimal DFA.
Match that minimised DFA against above file of sequences.

Compare the speed of matching each NFA, DFA, and minimal DFA.

Approximate matching (5p): Use the example from lecture slides and create new automata that can now recognise the above regular expressions with errors (edit distance 1 and 2).

Create the new 0, 1 and 2 error enabling NFA, and match it against the same file. Report # of counts and some example matches with exactly 0, 1 and 2 errors.

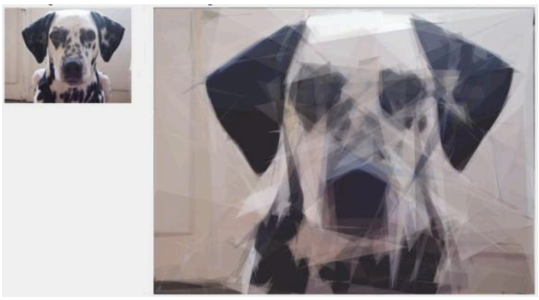
Create the respective DFA; Minimise that DFA. Report the matches by that DFA.

Compare matching the same regexp with standard libraries. Why and when you would prefer your own implementations (e.g. what functionality could you add?). You can experiment with larger alphabet sequences, etc.

EX3 (15p): **Optimisation** - regenerate input image(s) by algorithmic approaches with a (very) small number of features only, using some optimisation techniques. Report nr of features and approximately how much space such compact representation would take.

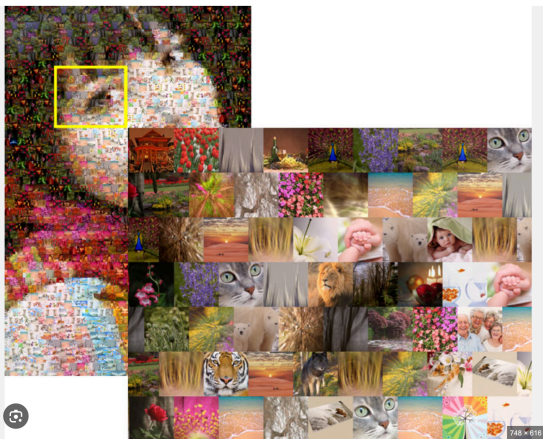
Select some attractive images or photos that you like and re-create them by some optimisation technique with a small nr of features. Choose one approach you like and implement that in code. Note that different methods below may require different images to start with. Make results “artsy”.

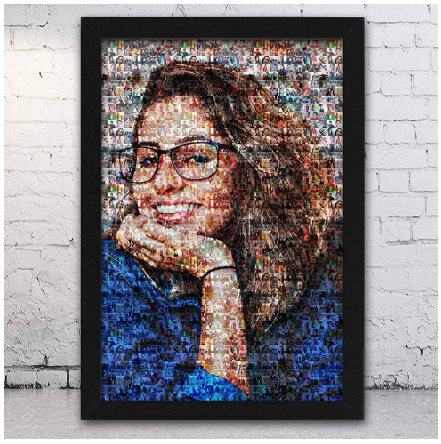
Few geometric elements: E.g. the evolutionary algorithm (GA, DE) regenerating images by a small nr. of geometric elements (triangles, circles, rectangles … with colors of variable transparencies) - few elements



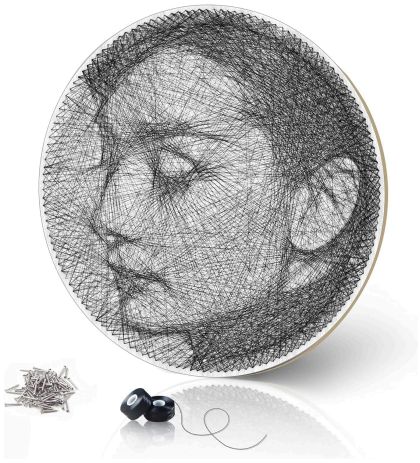
On the totally kitsch side of things, a dog, polygonated

Photo mosaic: Represent a larger photo by small thumbnail images from own image library, selecting for similar patterns greedily or using some optimisation (e.g. avoid using same photos repeatedly)





String art: simulate a threading of a line (BW line) to generate an image



Voronoi art (a few points) ; or Pointillism - few points of variable sizes;

You can also select the points combined with the TSP or MST across the chosen points.

Perhaps there are even more options available…

Choose one method (that you can relate to the best)**:**

Motivate your choice (curiosity)

Explain your solution / idea

Report the most essential part of your algorithm as pseudocode or function in the report (the rest remains in the code).

Indicate in the report the **computational complexity** and **compute time** implications.

Provide 2-3 illustrations of the results of your piece of art - in report; but also as files (zip? link?)

How small can you make your images using these new minimalistic representations?

Reflect on the results - are you happy with the results - algorithmically and artistically?